# On Designing the Method of Syntax Descriptions for Logics

Toshiro Minami*, Hajime Sawamura*, Kyoko Ohashi**, and Kaoru Yokota**

## 1. Introduction

This paper deals with the feature of describing the syntax of logical expressions for the general-purpose (i.e. logic-independent) reasoning assistant system EUODHILOS[2]. It is a system which assists us in reasoning in a variety of universes of discourse. This is why we call it general-purpose, and it follows that we have to define the logic to be dealt with at the beginning. It also gives us supports in the proofs of the theorems in the defined logic.

In the system, a logic consists of language and derivation systems. The former consists of making fonts for new symbols and of describing the syntax of logical expressions. The latter consists of axioms with two kinds of rules; inference and rewriting rules. In the current version, the style of inference rules is fixed like that of natural deduction. Rewriting rules are also given similarly. Therefore we have to take care mainly on the syntax description.

At the rest of this paper, we show how the syntax is described in the current version and some ideas to improve it.

## 2. Description Method

On designing the description method for logical expressions, we considered it from the following two points of view:

- Easiness for description
  Since the syntax is described by users, it must be simple and easily understandable.
- Expressiveness
  Some logics may be complex and could not be expressed in a context-free grammar. So context-sensitive conditions should be expressed in the language.

As the description method which satisfies these conditions, we have chosen DCG (Definite Clause Grammar)[3]. It is based on the context-free grammar style of description, augmented with argument attaching and prolog predicate call features. Using these features, we can express the context-sensitive restrictions to the syntax. In addition to these, it also has a well-known parsing algorithm named BUP[1].

In order to make more convenient to the system, we have modified DCG by augmenting the following two features.

- Operator declaration
- "Or"-notation

The former is added for automated generations of a parser and an unparser from the description. The parser is used to translate from the expression given by the user as a string to the internal representation which is used in the system, and the unparser translates in the opposite direction. An example of description is shown in Figure 1.

## 3. Some Improvements on the Current Version

As noted in the previous section, the DCG-based description method has many advantages. Though it still remains some problems for our purpose. For example:

*International Institute for Advanced Study of Social Information Science(IIAS-SIS), FUJITSU LIMITED, 140 Miyamoto, Numazu, 410-03, JAPAN.
Email: minami%iias.fujitsu.co.jp@uunet.uu.net

**Software Development Lab., FUJITSU LABORATORIES LIMITED, 1015 Kamikodanaka Nakahara-ku, Kawasaki, 211, JAPAN

```
formula --> formula1, ''⇒'', formula1;
formula --> formula1;
formula1 --> formula1, ''∨'', formula2;
formula1 --> formula2;
formula2 --> formula2, ''∧'', formula3;
formula2 --> formula3;
formula3 --> ''('', formula, '')'';
formula3 --> ''¬'', formula3;
formula3 --> basic_formula;
                  ⋮
operator
  ''⇒''; ''∨''; ''∧''; ''¬'';
```

Figure 1: An example description in the current version.

(i) Operator precidence are described both in the syntax description and in the operator declarartion.

(ii) It is impossible to express the functions which have arbitrary number of arguments.

(iii) In the general framework of DCG, any prolog predicates can be invoked during parsing. This makes it difficult to analyze the description. So we want to restrict the description style so that the description becomes clearer.

To solve the the first problem, we can take the way of omitting the description of operator precidence in the syntax description part. So the user describe the operator recidence only in the operator declaration part. For the second one, we may introduce the "..."-notation. And for the last one, by introducing the features such as automated parser/unparser generation, the necessity of calling prolog predicates will decrease.

As the result of using the first one, (and also using the "or"-notation,) the desciption in Figure 1 becomes simpler as in Figure 2.

```
formula --> formula, ''⇒'', formula |
  formula, ''∨'', formula |
  formula, ''∧'', formula |
  ''¬'', formula |
  basic_formula;
basic_formula --> ''('', formula, '')''|
  ...
                  ⋮
operator
  ''⇒''; ''∨'':left; ''∧'':left;
  ''¬'';
```

Figure 2: An simplified description for the example in Figure 1.

## 4. Further Improvements

For the future work, we present here an issue about learning feature in syntax description. In the process of reasoning, the user may change the descriptions many times. In the current description style, even when adding a new notation, one has to see all the descriptions and to remake all of them. If the system makes some modifications automatically, the user may teach the system by saying to add such and such a notation. The modification becomes much easier than that in the current situation.

## References


[1] Matsumoto, Y., et al.: BUP: A Bottom-Up Parser Embedded in Prolog, New Generation computing 1, pp.145–158, 1983.

[2] Minami, T., Sawamura, H., Satoh, K., and Tsuchiya, K.: EUODHILOS: A General-Purpose Reasoning Assistant System–Concept and Implementaion–, IIAS Research Report No. 84, FUJITSU LIMITED, 1988.

[3] Pereira, F.C.N., et al.: Definite Clause Grammars for Language Analysis–A Survey of the Formalism and a Comparison with Augmented Transition Networks, AI Journal 13, pp.231–278, 1980.



This work is a part of the major research and development of FGCS project conducted under the program set up by MITI.